Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2014, Article ID 971587, 17 pages
http://dx.doi.org/10.1155/2014/971587

Research Article

Estimating Energy Savings in Smart Street Lighting by Using an Adaptive Control System

Soledad Escolar,[1] Jesús Carretero,[1] Maria-Cristina Marinescu,[2] and Stefano Chessa[3]

[1] *Computer Science Department, University Carlos III of Madrid, Leganés, 28911 Madrid, Spain*
[2] *Computer Science Department, University of Pisa and ISTI-CNR, 56127 Pisa, Italy*
[3] *CASE Department, Barcelona Supercomputing Center, 08034 Barcelona, Spain*

Correspondence should be addressed to Soledad Escolar; sescolar@arcos.inf.uc3m.es

Received 5 July 2013; Revised 16 October 2013; Accepted 28 October 2013; Published 8 May 2014

Academic Editor: Yu Gu

The driving force behind the smart city initiative is to offer better, more specialized services which can improve the quality of life of the citizens while promoting sustainability. To achieve both of these apparently competing goals, services must be increasingly autonomous and continuously adaptive to changes in their environment and the information coming from other services. In this paper we focus on smart lighting, a relevant application domain for which we propose an intelligent street light control system based on adaptive behavior rules. We evaluate our approach by using a simulator which combines wireless sensor networks and belief-desire-intention (BDI) agents to enable a precise simulation of both the city infrastructure and the adaptive behavior that it implements. The results reveal energy savings of close to 35% when the lighting system implements an adaptive behavior as opposed to a rigid, predefined behavior.

1. Introduction

Providing an appropriate level of city lighting in public spaces is an important issue both for citizens and city councils. Most cities have strategic plans that specify how lighting is to be provided, as well as the intensity levels required for each area. The goals addressed by these plans must be agreed upon among politicians, citizen associations, businesses, and other city stakeholders and usually reflect trade-offs among the goals of the different actors. On one hand, city councils (and possibly environmental agencies) want to provide a sufficient service without wasting energy and money; on the other hand, the rest of the stakeholders ask for comprehensive lighting that can assure safety and create a psychologically positive social environment.

City lighting [1] has always been a major concern, as it represents 10% to 20% of the electricity use in most countries—sometimes more in developing countries. One of the measures that are most favoured by city councils involves the replacement of the emission devices with new ones based on LED technology [1–4]. But even without investing into replacing less efficient devices, a significant amount of energy can be saved by a more intelligent control of the lighting system [5–9]. This is a trend towards highly sustainable systems that are quickly gaining ground. References [10, 11] describe smart street light controllers with dual function of timing control and automatic photoelectric control to save energy in lampposts.

The drawback when evaluating the impact of such approaches is that the estimation of the energy savings entailed by this kind of lighting systems is not as straightforward as calculating the benefit of lamppost replacement. This is due to the potential complexity of the control system and the circumstances that may affect its behavior. Having tools that can precisely simulate the smart cities scenarios and estimate the potential savings in the lampposts with reasonable accuracy before committing to real deployment is fundamental to effectively support such complex processes.

In this paper we introduce a simulator for intelligent street light control systems. This simulator allows users to evaluate the energy efficiency of different public lighting configurations before deciding on a solution and implementing it on

site. Our solution assumes a wireless sensor network (WSN) platform and multiphase lamppost functionality based on LEDs, and it is based on an agent-based approach. The smart city scenario we are simulating involves a set of intelligent devices—in the Internet of Things (IoT) terminology *things*—which communicate and coordinate their actions to achieve intelligent street lighting. The devices are installed on the lampposts within the neighborhood that is being monitored and, additionally, in other strategic positions within this area. They have the ability to sense the environment and share their views with each other. This enables them to construct a global picture of the site and make informed decisions about lighting depending on the real-time necessities at their location. The purpose of the system we are simulating is to use the street lighting in an energy efficient manner while guaranteeing service as needed to avoid a negative social impact.

The lamppost control devices in our scenario execute applications that are aware of their context and have the ability to continuously and dynamically adapt their behavior in response to changes in the environment (e.g., ambient light, movement in the environment, and behavior of other communicating devices). In principle the application running on a lamppost may be different from the application running on any other; in practice nevertheless we expect that these applications fit the roles found in WSNs: data collector, cluster head, or sink. An application consists of a set of rules which describe all the possible lamppost behaviors under every relevant set of conditions. Some of these conditions may refer to local phenomena (e.g., values sensed by the device); others may implicitly specify more global properties (e.g., via values passed in messages from other devices). When a change which is specified as a rule condition occurs, the device may choose to execute a possibly new behavior that is more suitable and efficient in managing its new state. For instance, the application could define a rule to reduce the intensity of a street light if additional nearby lighting is detected. This continuous adaptation to the actual conditions provides flexibility to the applications running on the devices and ultimately results in a sustainable usage of the smart city infrastructure. Decisions are made at two levels: (1) local, where the devices react only based on the information that they possess from their environment and (2) globally distributed, where the devices interact and cooperate with other devices to react to changes in the environment and make global decisions that control the part of the system that they are responsible for.

The remainder of this paper is organized as follows. Section 2 reviews the related work in smart cities and intelligent street light control systems. Section 3 presents the background in autonomic computing systems and the novel approaches that combine WSNs and agents-based systems. In Section 4 we present the system model for self-adaptive applications. Section 5 describes a smart city scenario where lampposts are equipped with devices able to react to the changes in their environment with the purpose of saving energy in the lampposts. Section 6 presents the simulation results and Section 7 discusses the conclusions and future work.

2. Related Work

The European Research Cluster (IERC) on the Internet of Things (IoT) [12] has recently identified the most relevant application areas and has grouped them in twelve different vertical themes; one of them is related to *smart cities*. The concept of smart city is still to be precisely defined; however, the ultimate goal of a smart city is to improve the quality of life of the population while guaranteeing a sustainable development.

About 50% of the world population lives in cities and it is estimated that this number will continue to grow to reach 70% by 2050 [12]. The resources demanded by citizens—energy, water, land, and so forth—will also grow at least linearly with the population size; however, it is difficult to see how the supply could follow the same growth curve. This fact imposes severe requirements to managing the basic resources efficiently and sustainably. New technologies have a tremendous potential to monitor and analyze resource-consuming processes and to bring added value to the services that a city provides. The infrastructure of the smart city—composed of millions of object instances and heterogeneous devices—must act as the pervasive technology [13] which, managed appropriately, can increase the knowledge that citizens, public organizations, and businesses have of their environment and can help making smart decisions with the involvement of the community.

One of the scopes that IERC identifies for smart cities is *smart lighting* [12], which is also the problem addressed in this paper. The design of control strategies meant to match the emitted light level to the actual needs can lead to a significant reduction in energy costs and, in turn, to an efficiency improvement. One of the popular strategies that city councils carry out is replacing old, expensive lights by low-cost and low-CO_2 emissions devices such as the LED-based devices [2]. According to [1], using LED technology can improve energy efficiency by up to a factor of five without altering light intensity levels and with a relatively low effort. The work described in [3] analyzes the effect of using LED street lights from technical and economic perspectives to indicate savings of $100,000 per year. Similarly, a study for the city of Dublin [4] shows that currently the township spends almost $250,000 a year for a street lighting system with 2,019 lights, 30% of them (the more expensive) mercury vapor lights. They predict that replacing these with more efficient LED-based lights would save about $15,000 per year.

In addition to replacing less efficient devices, many cities are taking measures that balance citizen satisfaction, expenses, and energy efficiency to achieve a more sustainable lighting system. In [14] the authors present a goal programming (GP) model designed to capture multiple objectives involved in sustainable energy-environment management in urban areas. In [15] the authors show a model for rational sustainable development in Vilnius. Jabareen [16] identified sustainable *urban forms* and their design concepts and equipments. In addition, he addresses the question of whether certain urban forms contribute more than others to sustainability.

The issue of urban energy consumption is not only relevant for street lighting but also for public buildings. In Europe it is estimated that the amount of electric energy consumed in illuminating the interiors of medium and large public buildings is about 40% [5] of the total electrical energy used. This study identifies different kinds of buildings (e.g., offices, hospitals, hotels, and restaurants) and defines a light control system that regulates the emission based mainly on two parameters: daylight and occupancy. The evaluation reveals that the system results in energy savings of 25% in industrial and commercial sectors and up to 45% in tertiary and educational sectors. This approach uses WSNs to monitor the environmental conditions. The intelligent street light control system proposed in [6] relies on LED-based lampposts and wireless sensors. Based on a predefined schedule and the sensor information about weather conditions and detection of pedestrians, the system adapts the level of lighting of the lampposts to achieve energy savings between 30% and 40%. In [7] the authors define a control system for an indoor scenario by using a WSN and address the problem of the energy consumption in the sensor nodes by proposing an energy-aware communication protocol which increases the lifetimes of the sensor nodes by 20%. Reference [8] analyzes distributed and centralized models for building environmental control systems. The authors use simulation to evaluate and compare the two models in terms of latency, performance, packet loss, and computational complexity. They demonstrate that, in general, a distributed model behaves better than a centralized model.

The works cited above define static behaviors that allow the sensors nodes to regulate the light level of each lamppost. These approaches lack the degree of autonomy and (self-) adaptation that results from continuously reasoning about the changes that occur in the environment of the sensor node (be it purely local or the result of communicating with other nodes). In fact, [12] recognizes that there exists "a lack of research on how to adapt and tailor existing research on *autonomic computing* to the specific characteristics of IoT, such as high dynamicity and distribution, real-time nature resource constraints, and lossy environments." Agents-based systems have been identified as one of the most suitable technologies that could potentially improve the flexibility, robustness, and autonomy [17] of WSN-controlled systems. Reference [9] presents the design of an multiagent systems (MAS) that uses a WSN to control the lighting of commercial spaces; it additionally discusses the issues that are important in this type of systems. The authors organize the global sensor network in different subnets according to the preferences of the users. Each subnet consists of a set of sensor nodes, each of them executing a single agent. This is still preliminary work and the authors do not present an evaluation of their approach.

The work we present in this paper introduces an intelligent street light control system that combines three elements: (1) LED-based lampposts, (2) wireless sensor networks, and (3) agents. Our objective is estimating the energy savings in the lampposts as a result of replacing the fixed behavior of the sensor nodes with an adaptive behavior. This paper contributes to the state-of-the-art in two ways: (1) it describes a way of modeling and simulating the features of smart cities scenarios such that we can estimate the potential energy savings before deployment, and (2) it proposes an adaptive control system that can be implemented by the devices that compose the smart city infrastructure and which ensures sustainability without sacrificing the level of services offered to the citizens.

3. Background

Autonomic computing [18] refers to the self-managing characteristics of distributed computing resources, which have the ability to adapt to changes while hiding the intrinsic complexity from the users. Several research groups have focused on autonomy and self-adaptivity for embedded systems [19–22]. The components of an autonomic computing system should be able to independently react to their environment and interact between them to achieve their goals. One of the paradigms which most naturally fits this model is the agent-based approach. Agents have the ability to manage their local knowledge, sense their environment, and share knowledge with other agents [23]. They are continuously reacting to changes in their environment and making decisions that can lead to changes in their behavior. There exist different classes of agents depending on their reasoning capability. We are focusing on a particular class of agents called BDI (belief-desire-intention) agents, which have a very powerful reasoning engine but generally consume a lot of computing resources. BDI [24] agents are defined by an internal state that consists of a set of beliefs (information that the agent has about the world), desires (a set of goals), and actions (a set of plans to achieve the goals). The intentions represent the current state of progress of the actions started by the agent, following some plan that has not yet been completed. Programming languages such as AgentSpeak(L) [25] provide a way of programing BDI agents where beliefs, desires, and intentions are not explicit formulas in the language; instead it uses events, contexts, goals, and actions. In AgentSpeak(L) a plan is generally defined by a rule of the form `event: context ← list_to_do`, where `list_to_do` is a set of actions and/or goals to be achieved by the agent when the `event` occurs in the specified `context`. An event is triggered when the agent acquires a new goal or detects a change in the environment (i.e., addition or deletion of goals/beliefs). The context specifies the beliefs that must hold when the plan is triggered. As an example of event, consider rule 3 in Reaction Plan 1. This rule says that when an event *VAL_SOLAR_LIGHT(s)* is triggered as result of sampling the environment, if the event parameter *s* is greater than a constant value *TH_DAY*, then the plan body on the right hand side of the arrow executes; otherwise, the plan is discarded for execution in this step.

Jason [26] provides a simulation framework based on BDI agents and AgentSpeak(L); it implements a metalanguage on top of AgentSpeak(L) and builds an interpreter for this metalanguage. Each agent in Jason behaves as a finite state machine but may also implement additional functions (called

Let T be the number of times an object must be sensed to consider it present
Let *TH_DAY*, *TH_VOLT*, *TH_ENV* be the thresholds for the solar light, energy, and environmental light
Let *Presence* be the reading of the presence sensor (initially Presence == null)
Let *PresenceStable* = *False* be the value to indicate that the value of *Presence* is stable or not
Let *PresenceCounter* = 0 be the number of consecutive times that the presence sensor detects the same value
Let *Nighttime*, *EnvLight* be the reading of the solar and environmental light sensors, respectively

% **Block A**: Local behavior—computation
(1) **SAMPLING_TIMER**: ← **Read**(SOLAR_LIGHT), **Read**(ENV_LIGHT), **Read**(LIGHT_LAMP), **Read**(VOLTAGE), **Read**(PRESENCE_SENSOR)
(2) **VAL_SOLAR_LIGHT(s)**: (s ≤ TH_DAY) ← set_lamp(HIGH), set_nighttime(True)
(3) **VAL_SOLAR_LIGHT(s)**: (s > TH_DAY) ← set_lamp(OFF), set_nighttime(False)

% Rules 4–7 detect continuous absence or presence of obstacles for the past T time intervals
(4) **VAL_PRESENCE(p)**: (Presence == null) ← set_presence(p), set_presence_stable(False), reset_presence_counter()
(5) **VAL_PRESENCE(p)**: (Presence != null) ∧ (p != Presence) ← set_presence(null)
(6) **VAL_PRESENCE(p)**: (Presence != null) ∧ (p == Presence) ∧ (PresenceCounter ≤ T) ← inc_presence_counter()
(7) **VAL_PRESENCE(p)**: (Presence != null) ∧ (p == Presence) ∧ (PresenceCounter > T) ← set_presence_stable(True)
(8) **VAL_ENV_LIGHT(e)**: ← EnvLight = e
(9) **true**: (PresenceStable) ∧ (Nighttime) ∧ (~Presence) ← set_lamp(LOW)
(10) **true**: (PresenceStable) ∧ (Nighttime) ∧ (Presence) ∧ (EnvLight > TH_ENV) ← set_lamp(MEDIUM)

% **Block B**: Global behavior—communication
(11) **SENDING_DATA**: (Nighttime) ←**SENSOR_MSG**(Sender = id_sensor, Receiver = id_cluster_head, *sensorMeasurements*)
(12) **CLUSTER_SMSG(Msg):** (Msg[SolLight] ≤ TH_DAY) ← set_lamp(Msg[ActivityLevel])
(13) **CLUSTER_QMSG(Query)**: ←**SENSOR_QMSG**(Sender = id_sensor, Receiver = Msg[Sender], PresenceStable, ~Presence)

REACTIONPLAN 1: Behaviors for data collectors.

internal actions). Jason also supports features such as multithreading and a Java-based graphical monitoring tool.

Several more recent works have combined sensor networks and agents. These model each sensor node as an agent programmed using an agent-based language. This abstraction enables the sensors to act autonomously and react to the environment. A survey of these simulation tools is presented in [27]. SAMSON [27] is one such tool which provides a framework for WSNs on top of Jason and models every sensor node as an AgentSpeak(L) agent. SAMSON supplies every agent with the radio and energy models of the TMote Sky [28] platform and enables the simulation of sensor nodes that behave as BDI agents. The framework can measure the energy consumption and the number of packets received and transmitted by each sensor node.

4. System Model

We consider a WSN consisting of N sensor nodes. Every node k ($0 \le k < N$) is provided with a possibly different set of behaviors (or *reaction plans*) $\mathrm{RP}_k = \{P_k^1, P_k^2, \ldots, P_k^n\}$. A sensor node usually plays one of the roles of data collector, cluster head, or sink. It is commonly the case that each set of nodes fulfilling the same role implement the same behavior, although different behaviors are allowed. In principle every data collector node is associated with one lamppost; cluster heads and sinks may be deployed in different locations as well as on lampposts.

A behavior P_k^i ($1 \le i \le n$) is defined by the triple $\langle e_k^i, C_k^i, A_k^i \rangle$, where e_k^i is the triggering event and C_k^i is the set of conditions that must hold for P_k^i to be eligible for execution. When P_k^i is chosen for execution, it will execute the subset of tasks A_k^i from the set of all tasks $\Theta = \{A_0, \ldots, A_\alpha\}$. The mechanism by which a behavior is elected for execution between many eligible behaviors and the semantics of executing the corresponding set of actions are explained in Section 4.1. During their lifetime the sensor nodes react to changes in their environment and to messages received from other sensor nodes by choosing the best course of action out of the behaviors defined by RP_k. At configuration time, every node k is assigned an initial behavior $P_k^0 = \langle e_k^0, C_k^0, A_k^0 \rangle$.

Even if the sensor lifetime is an important issue—one which we have already addressed in [29, 30]—our purpose in this paper is to minimize the energy waste of the lighting system as provided by the lampposts. Since data collector nodes are attached to the lampposts, they can be directly powered from the power grid. Additionally, the energy required to power on the sensors is typically much lower than the energy required to power up the lampposts; we have therefore disregarded this aspect in the paper and we focus exclusively on the management of the lampposts.

4.1. The Execution Semantics. The execution of any task in A_k^i can generate a set of *internal* events. Consider for instance a task that measures the light emitted by a sensor node. Since the latency of the hardware when performing this operation cannot be neglected, there is a lapse of time between the instant when the operation is started and when it ends. When the operation finishes the hardware generates an internal event (e.g., lightDone()) to announce that the data is available. A sensor node may also observe *external* events which are generated by the environment rather than being a result of

the execution of some of its tasks. Examples of external events are the reception of a message from another sensor node, an alarm that goes off when a value exceeds some threshold, and so on.

Both internal and external events that occur in a node k may have the role of a triggering event for some behavior. External events are enqueued in the *event queue* $E^k[]$ in the order in which they occur. Meanwhile, a newly generated internal event will be processed immediately. We can think of an internal event as being conceptually processed as part of processing the external event whose associated task generated it.

Our simulations execute on top of a language based on AgentSpeak(L) and therefore adopts its execution semantics. When an event is observed, every behavior P_k^i triggered by this event must evaluate its triggering condition C_k^i. If the condition evaluates to false, P_k^i is discarded as a candidate for execution; otherwise, it becomes eligible. AgentSpeak(L) implements a selection function S_O which chooses one of the eligible behaviors for execution.

The lifetime of a sensor node is discretized in reasoning cycles of a given duration. This duration is chosen to be the maximum of the worst-case estimated times that it takes for the node to process the external events (which it recognizes) to quiescence. Given that this time depends on the set of behaviors implemented by each sensor node, the reasoning cycle may have a different value for each node. Algorithm 1 describes the infinite reaction loop which is executed locally and independently on each sensor node. Every (local) reasoning cycle the sensor node removes an external event from its event queue $E^k[]$ by using the AgentSpeak(L) operator S_E; then it chooses the next behavior to execute by using the operator S_O over the set of eligible behaviors. Remember that executing an action A_k^j may generate (a set of) internal events $\{ie\}$. These are processed as part of the execution of the external event rather than being queued explicitly. The last line of Algorithm 1 (`Execute` A_k^j) may therefore include the execution of additional sets of tasks corresponding to behaviors triggered by the events in $\{ie\}$.

The reason we can only compute an approximation of the event processing time is that we cannot know for sure which other behaviors will be executed as a result of tasks generating internal events. For instance, events are allowed to have parameters and the triggering condition may depend on the values taken by these parameters. If a value passed as a parameter to an internal event can be written by the behavior which generated the internal event, we cannot know in advance which behavior this will trigger if any. Conditional generation of internal events is another case in which the set of behaviors executed as part of processing an external event is not known at static time.

4.2. Energy Estimation. Every behavior P_k^i has a cost W_k^i which depends on the energy cost per unit of time, and that can be expressed as

$$W_k^i = \sum_{j=1\cdots A_k^i} V \times t_j \times I_j, \tag{1}$$

```
Let RP_k be the set of behaviors defined for node k
loop
  j = S_E(E^k[]) // Select some event in E^k[]
  for all
  P_k^i = ⟨e_k^i, C_k^i, A_k^i⟩ in RP_k s.t. j == e_k^i do
    holds = true
    for all c ∈ C_k^i do
      if eval(c) == false then
        holds = false
    if holds == true then
      Insert i in Eligible_k
  P_k^j = S_O(Eligible_k)
  Execute A_k^j
```

Algorithm 1: Adaptation algorithm.

where t_j is the execution time for task $j \in A_k^i$, V is the voltage required by the device to work properly, and I_j is the current draw required to execute this task. As we explained in the previous section, the processing of an external event to completion may result in the processing of additional internal events. The estimation of the energy it takes to execute the entire set of behaviors triggered by an external event is therefore a worst-case estimation, just like the calculation of the reasoning cycle time.

5. Application for Smart Lighting

We consider a WSN composed of N sensor nodes that execute an adaptive application which models a smart lighting system using the approach described in Section 4. The WSN is deployed on a street with lampposts emitting variable light intensity. The sensor nodes may take one of three roles: (1) data collectors which are attached to lampposts and have a fixed position, (2) cluster heads which are deployed in strategic positions within the WSN to facilitate the grouping of several sensors for fault-tolerance, forwarding, or aggregation purposes, and (3) the sink which is the target sensor node which collects all the data from the WSN. Data collectors are equipped with five transducers: `LIGHT_LAMP` for the intensity of the light generated by the lamppost, `ENV_LIGHT` for the intensity of the light in the environment, `SOLAR_LIGHT` for perceiving solar light, `PRESENCE_SENSOR` to detect the presence of obstacles in the environment, and `VOLTAGE` to measure the remaining battery voltage. Each data collector is also equipped with an actuator `ACT_LAMP` which may be used to regulate the intensity of the light emitted by the lamppost. The cluster heads only have the transducer `VOLTAGE`.

A lamppost can be in one of four states: `OFF`, `LOW`, `MEDIUM`, and `HIGH`. Our smart lighting application allows for adjusting the level of light according to a series of conditions which we describe below. The energy consumption of a lamppost is proportional to the intensity of the light it produces. The goal of this application is to reduce the global energy consumption for street lighting while providing an adequate degree of service.

5.1. Specifying Behavior by means of Rules. In our system, the sensor nodes (data collectors, cluster heads, and the sink) execute what we call a reaction plan. The name is meant to capture the idea that our approach is best suited for specifying reactive systems. If the behavior of two sensor nodes is different (be they of different types—such as data collectors, cluster heads, and sink—or of the same type) then their reaction plans are also different. In general all nodes of the same type will have the same behavior; the developer will only have to specify three reaction plans, one per type.

Different from the traditional syntax used to implement algorithms for sensor nodes, we adopt a rule-based language. Each behavior included in the reaction plan is represented as a rule with a triggering event (possibly parameterized), a context, and a body. The body is the only component of the rule which is not optional. We use a similar notation to that of AgentSpeak(L), which was already introduced in Section 3:

`[event] ([parameters]): [context]` ← `body`,

where event can be both an internal or an external event, context is the set of conditions that must hold for the rule to be executable, and body specifies the actions to be executed. The execution semantics that we use was described in Section 4.1.

The choice of a specification approach based on rules is an important decision. While a specification using imperative `if-then-else` constructs may traditionally be more common, it is not equivalent to a set of rules. First, rules are triggered by events and therefore naturally specify reactive behavior such as that of WSNs. The condition of an `if-then-else` construct is evaluated only when the code is explicitly invoked. To find the behavior that will apply the execution must evaluate all `if-then-else` conditions that textually precede it. No freedom is allowed to choose between multiple behaviors that would be eligible in a rule-based specification. Once passed the somewhat unfamiliar syntax, the developer will find that composition of behavior is considerably easier in our approach because it is implicitly done by the compiler or the runtime system. This approach does not involve extraneous code compared to other approaches that use more traditional specification languages.

It is important to note that our syntax does not distinguish computation from communication. Communication between sensor nodes is done by generating and consuming external events. From the point of view of the specification, it is irrelevant whether event triggering is due to sensing a change in the local environment or a message received from another node. This feature simplifies the language and makes applications conceptually more uniform and easy to modify and reuse. It is the decision of the developer whether the reaction plans implement global behaviors that are distributed or centralized; coordination and synchronization between nodes are achieved by generating external events and appropriately setting the values of their parameters.

We use different fonts to represent different elements during the explanation of the three reaction plans. We use typewriter font to indicate the names of transducers and italic font to represent both variables and events.

5.2. Reaction Plans. Reaction Plans 1, 2, and 3 describe the set of behaviors implemented by the data collector, cluster head, and sink type nodes in our WSN.

Data Collector Nodes. The reaction plan for this type of nodes is specified in Reaction Plan 1. As components of a data collector node, the `SOLAR_LIGHT`, `ENV_LIGHT`, and `PRESENCE_SENSOR` transducers generate the events *VAL_SOLAR_LIGHT, VAL_ENV_LIGHT*, and *VAL_PRESENCE*; we do not explicitly show how this happens but rather assume that the environment produces them. Additionally, the periodic external events *SAMPLING_TIMER* and *SENDING_DATA* are generated by two timers. Note that local/global refers to physical location, whereas internal/external refers to whether an event was produced by an executing task with the intention of being processed immediately or whether this was produced by the environment external to the executing application to be queued and processed at some point in the future. *SENDING_DATA* triggers an external event *SENSOR_MSG(m)* which is processed by the corresponding cluster head, and it therefore specifies global, internode behavior (one can think of this as a communication component of the application). The rest of the events trigger purely local behavior. The data collector nodes also implement internode communication behavior as a reaction to receiving messages from the corresponding cluster heads; we consider the receipt of a message to be an external event.

The rules that control the intensity of the light emitted by a lamppost are described next. During daytime conditions the lamppost is turned off; at nighttime, the lamppost is turned on either `LOW`, `MEDIUM`, or `HIGH`, depending on a series of conditions. The value of a sample taken by the `SOLAR_LIGHT` transducer indicates whether the light conditions are those of daytime or nighttime depending on whether this value is above *TH_DAY*. This value is passed as a parameter to the *VAL_SOLAR_LIGHT* event, which is generated when the measurement is done. For instance, rule 2 of Reaction Plan 1 reads as follows: when the external event *VAL_SOLAR_LIGHT(s)* is observed by a data collector node, if the value of the event parameter *s* is not greater than a fixed threshold *TH_DAY*, then the node sets the actuator `ACT_LAMP` such that the lamppost produces light with `HIGH` intensity and additionally sets the local variable *Nighttime* to *True*. When a lamppost is turned on its level of intensity is set by default to `HIGH`. We can always set multiple finer thresholds to distinguish between different time frames (e.g., sunrise to noon and noon to sunset).

We configure a data collector *k* to wake up every sampling time period and sample each one of its five transducers; rule 1 implements this behavior. To be precise, the data collector does not necessarily sample its environment at equal time intervals. *SAMPLING_TIMER* events are periodically generated and queued. As we described in the previous section, there may be multiple events in the queue, and the processing of an event may be delayed until after the processing of other previously queued events.

At nighttime, periodic *SENDING_DATA* events trigger the sending of a message containing the sampled data values to the corresponding cluster head; rule 11 describes this

Let DC be the number of data collectors within a cluster dominated by the cluster head
Let *TH_VOLT* be the threshold value for the energy sensor
Let *DCCounter* = 0 be the number of different data collectors within a cluster that sent a message with their readings
Let *NoPresenceCounter* = 0 be the number of different data collectors within a cluster that did not detect presence
Let *Voltage* be the reading of the energy sensor
Let *DataCollectorVector*[] be a vector that contains the identifiers of DC data collectors
Let *PresenceVector*[] be a vector that contains the values of valid presence detected for DC data collectors

% **Block A**: Local behavior—computation
(1) **SAMPLING_TIMER**: ←**Read**(VOLTAGE)
(2) **VAL_VOLTAGE(v)**: ← Voltage = *v*
% **Block B**: Global behavior—communication
(3) **CLUSTER_MSG(Msg)**: ←**CLUSTER_MSG**(Msg)
% Rule 4 forwards each received message from the data collectors to the sink and queries the data collectors for information about the continuous presence/absence of objects
(4) **SENSOR_MSG(Msg)**: (Msg[Voltage] > TH_VOLT) ← set_presence_vector(null), reset_nopresence_counter(),
CLUSTER_QMSG_n(DC, Sender = id_cluster_head, Receiver = DataCollectorVector[i]),
CLUSTER_MSG(Sender = Msg[Sender], Receiver = id_sink, *ParametersOfMsg(Msg)*)
% Rule 5 accumulates the values of the readings from DC data collectors in local variables
(5) **SENSOR_MSG(Msg)**: (Msg[Voltage] ≤ TH_VOLT) ∧ (DCCounter < DC) ← inc_dc_counter(), *computeAggregate(Msg)*
% Rule 6 computes the averages over all received values and sends the results as a unique message to the sink
(6) **SENSOR_MSG(Msg)**: (Msg[Voltage] ≤ TH_VOLT) ∧ (DCCounter==DC) ← reset_dc_counter(),
CLUSTER_MSG(Sender = Msg[Sender], Receiver = id_sink, *computeAverages*)
% Rule 7 receives the information about the continuous presence/absence of objects from those collectors which have not already sent this information
(7) **SENSOR_QMSG(Msg)**: (Msg[Sender]==DataCollectorVector[j]) ∧ (PresenceVector[j]==null) ←
inc_nopresence_counter(), PresenceVector[j] = Msg[Presence], *computeProducts(Msg)*
% Rule 8 is executed when DC data collectors detected stable absence; it turns off odd numbered lampposts and it turns LOW the rest
(8) **true**: (NoPresenceCounter == DCCounter) ∧ (ProdPresence == 1) ∧ (ProdPresenceStable == 1)←
CLUSTER_SMSG_n(floor(DC/2), *sendMessageEvenCollectors(LOW))*,
CLUSTER_SMSG_n(floor(DC/2)-1, *sendMessageOddCollectors(OFF))*
(9) **SENDING_DATA**: ←**SINK_MSG**(Sender = id_cluster_head, Receiver = id_sink, Voltage)

REACTIONPLAN 2: Behaviors for cluster heads.

Let *TH_VOLT* be the threshold value for the energy sensor
(1) **CLUSTER_MSG(Msg)**: (Msg[Voltage] ≤ TH_VOLT) ← Save(Msg), Print("Alert for battery in", Msg[Sender])
(2) **CLUSTER_MSG(Msg)**: (Msg[Voltage] > TH_VOLT) ← Save(Msg)
(3) **SINK_MSG(Msg)**: (Msg[Voltage] ≤ TH_VOLT) ← Save(Msg), Print("Alert for battery in", Msg[Sender])
(4) **SINK_MSG(Msg)**: (Msg[Voltage] > TH_VOLT) ← Save(Msg)

REACTIONPLAN 3: Behaviors for the sink.

behavior. Sending the information occurs via the only available mechanism: generating the corresponding event. In this case the event *SENSOR_MSG* contains several parameters; the id of the data collector node is specified as the sender and the corresponding cluster head as the receiver. Values measured by the transducers and local variable values follow in a predefined order; for the sake of readability we use a made-up variable name (*sensorMeasurements*) to stand for `Voltage`, `EnvLight`, `Nighttime`, `PresenceStable`, `Presence`, and `LightLamp`.

Lines 4–10 describe behavior rules which modify the intensity level of a lamppost at nighttime depending on the actual conditions at the site, specifically the detection of objects (or their absence) over a certain extent of time. `PRESENCE_SENSOR` returns 1 if an object is detected and 0 otherwise. If no object is detected for a number T of consecutive sampling periods we downgrade the intensity level to `LOW`. We choose to never turn `OFF` a whole section of lampposts at nighttime to avoid a negative social impact on pedestrians—which may choose to altogether avoid an apparently dark street. The variable *PresenceStable* indicates if the value of *Presence* returned by `PRESENCE_SENSOR` is stable; that is, all sensor readings over the last T consecutive sampling periods returned the same value, in which case *PresenceStable* takes the value *True* (see rule 7). If two consecutive readings of `PRESENCE_SENSOR` have a different value,

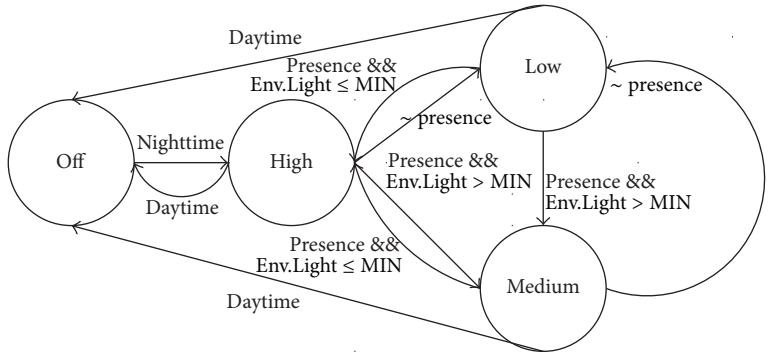

Figure 1: Transition diagram for the light intensity states.

rule 5 resets the variable *Presence* to `null`. Otherwise, rule 6 increments the variable *PresenceCounter* a maximum number of T times. If some object is detected continuously over the past T time samples and the environment light is above a certain threshold then the data collector node downgrades the intensity level to `MEDIUM`, as shown in rule 10. The continuous presence of an object is detected if *PresenceStable* has value *True* and the last reading was positive; that is, *Presence* has value 1. If no presence is detected for the past T time samples (*PresenceStable* is *True* and *Presence* is 0), then the data collector node downgrades the intensity level to `LOW`, as shown in rule 9. Figure 1 summarizes the transitions among different light intensity states for a lamppost. The states in the figure are labeled with the level of light intensity: `OFF`, `HIGH`, `MEDIUM`, and `LOW`; the labels on the arrows represent the conditions that must hold to transition from a state to another. To simplify the conditions, the variable *Presence* used in the figure means that presence was detected and it is also stable. To avoid excessive cluttering we do not show the self-loops corresponding to each state.

Lastly, a data collector may receive messages from its cluster head. In rule 12 the external event *CLUSTER_SMSG (Msg)* triggers an action which sets the activity level of the lamppost to the value indicated by the cluster head. Specifically, the rule reads as follows: when a *CLUSTER_SMSG* event with parameter *Msg* is processed, if the value of the field named *SolLight* of *Msg* is below the fixed threshold then the actuator `ACT_LAMP` is set to take the value passed in the field *ActivityLevel* of *Msg*. In rule 13 an event of type *CLUSTER_QMSG* prompts the data collector to send information about the continuous (aka stable) presence (or absence) of an object to the cluster head. This information is necessary to implement a cluster head policy which involves turning off subsets of lampposts. The data collector therefore needs to send two pieces of information: *PresenceStable* to capture stability of observation and *Presence* to reflect whether stability over the past T time intervals represents presence or absence of an object. In Reaction Plan 2 it will become apparent why the data collector sends as the fourth parameter the negated value of *Presence*.

Cluster Head Nodes. Reaction Plan 2 describes the set of behaviors implemented by a cluster head. A sensor node j acting as a cluster head ($j \neq k$) is configured to sample its battery level every sampling interval (rule 1) and send it to the sink every sending interval by generating an external event (rule 9) for energy management purposes.

When a message from one of its data collector is received (via the parameter of the *SENSOR_MSG* event) the cluster head checks the collector's voltage level. If this is greater than a certain threshold (*TH_VOLT*) then rule 4 is selected to execute; otherwise, rules 5 or 6 are selected. By generating the external *CLUSTER_MSG* event, rule 4 forwards the values it receives in the event parameter to the sink—if the battery level is above a given threshold. Notice that the sender field is set to the original data collector which is stored in `Msg[Sender]`, such that the sink may know the identity of the originating node for the transducer values. For legibility purposes we use the notation *ParametersOfMsg (Msg)* to stand for `Voltage = Msg[Voltage]`, `EnvLight = Msg[EnvLight]`, `SolLight = Msg[SolLight]`, `PresenceStable = Msg[PresenceStable]`, `Presence = Msg[Presence]`, and `LightLamp = Msg[LightLamp]`. This rule also generates events for the data collectors which request the values of the *PresenceStable* and *Presence* bits. Messages received from other cluster heads (via the *CLUSTER_MSG* event in rule 3) are simply forwarded regardless of the battery level; this implements the multihop message routing protocol.

If the cluster head receives values from the data collectors but its battery level is below the predefined threshold *TH_VOLT* then the node switches from forwarding every message received to an aggregator behavior. Rules 5-6 describe the process by which the cluster head aggregates the data proceeding from the last *DC* samples sent by data collectors in a unique message and forwards it to the sink. *DCCounter* is initially set to 0. To keep the example legible we aggregate *DC* readings regardless of whether they come from different *DC* data collectors or not (we assume that *DC* is the number of the data collectors that the cluster head is responsible for). By sending one single message instead of forwarding every one of them, the cluster head saves energy. We use the function *computeAggregate (Msg)* to stand for `add (SumVoltage, Msg[Voltage])`, `add (SumLightLamp, Msg[LightLamp])`, `add (SumSolarLight, Msg[SolLight])`, `add (SumEnvLight,`

Msg[EnvLight]), mul (ProdPresenceStable, Msg[PresenceStable]), and mul (ProdPresence, Msg[Presence]), where add(x,y) means x=x+y and mul(x,y) means x=x∗y. Similarly, for legibility purposes we use the notation *computeAverages* in rule 6 to stand for Voltage = SumVoltage/N, EnvLight = SumEnvLight/N, SolLight = SumSolarLight/N, ProdPresenceStable, ProdPresence, and LightLamp = SumLightLamp/N.

In rule 7, the presence information sent by the data collectors (as result of the request by the cluster head in rule 4) is received as the parameter of event *SENSOR_QMSG*. The rule collects the inverse of the values measured by the PRESENCE_SENSOR transducer of each lamppost in the cluster; it also collects the *PresenceStable* values. The function *computeProducts (Msg)* stands for mul (ProdPresence, Msg[Presence]), mul (ProdPresenceStable, Msg[PresenceStable]) and computes the values for the local variables *ProdPresence* and *ProdPresenceStable*. The triggering condition of the behavior in rule 7 reads as follows: for the data collector which corresponds to the sender of the *SENSOR_QMSG* event, check whether its corresponding *PresenceVector[j]* is null. This condition is true whenever no previous *SENSOR_QMSG* event has been received from this data collector in this counting round. *DataCollectorVector* and *PresenceVector* are two vectors local to the cluster head, of size equal to *DC*. *DataCollectorVector[i]* and *PresenceVector[i]* contain the identifier and the value of the presence bit for the *i*th data collector (*PresenceVector[i]* is initialized to null in rule 4 by the function *set_presence_vector(null)*).

In rule 8, if *DC* different collectors detect stable absence of objects then the cluster head generates an event for every other data collector that it is responsible for, such that these turn their light OFF; the rest of the lights turn down to LOW. The rule condition reads as follows: *NoPresenceCounter* == *DC* tests *DC* different collectors because rule 7 only triggers when *PresenceVector[j]* has not been previously set; that is, it only considers distinct collectors; *ProdPresenceStable* == *1* tests whether all readings were stable; *ProdPresence* == *1* tests whether all negated values of the presence sensor are 1; that is, all values of these sensors are 0. We decided to exchange the inverse values of the presence values (instead of the values themselves) to efficiently implement the test that every value returned is zero, that is, to implement the test that the stable condition captures absence, rather than presence, of an object.

Rules 4 and 8 use functions with the notation *eventname_n*(*h*, *parameters*) as syntactic sugar for the generation of *h* events *eventname*. For instance, in rule 4, a number (*DC*) of *CLUSTER_QMSG* events are generated, one for each data collector i ($0 \leq i < DC$). Rule 8 generates floor (DC/2) *CLUSTER_SMSG* events for the even numbered data collector nodes and floor (DC/2)-1 *CLUSTER_SMSG* events for the odd numbered nodes. For legibility purposes we use the function *sendMessageEvenCollectors (LOW)* that stands for the parameters (Sender = id_cluster_head, Receiver = DataCollectorVector[2∗i], and ActivityLevel = LOW) and *sendMessageOddCollectors (OFF)* that stands for the parameters (Sender = id_cluster_head, Receiver = DataCollectorVector[2∗i+1], and ActivityLevel = OFF). The algorithm works under the assumption that consecutively placed lampposts are numbered sequentially such that no large contiguous region is turned OFF at the same time. If another pattern is sought to choose the lampposts that stay on, this must be reflected in the way that rule 8 selects this set.

Sink Node. The reaction plan for this type of nodes is specified in Reaction Plan 3. The sink receives messages from the cluster heads and saves the message data. If the battery level is below *TH_VOLT* then the sink additionally prints an alert message. These messages may be of one of two types: *CLUSTER_MSG*—initiated at a data collector node and containing all the measurements—or *SINK_MSG*—initiated at a cluster head and containing only the energy measurement (since this is the only transducer available in a cluster head).

5.3. Complexity. The reaction plans that we are proposing use a set of rules to detect changes in the environment (including the events produced by other sensor nodes). While no change has been detected, none of the rules in the reacting plan can trigger. Detecting a change is represented by the processing of an event. When an event is processed it triggers the execution of a rule, which may in turn trigger other rules, either by generating internal events or by enabling rules with no triggering event. The processing of the event is considered finished when there are no more rules that can trigger in the current timestep. The key is that all of the rules that execute during a timestep are evaluated for execution in the state at the beginning of the timestep and they may not see the data changes produced by the others. This ensures convergence. Additionally, in every timestep the maximum number of rules which may execute is given by adding 1 to the sum of the number of rules triggered by internal events and the number of rules with no triggering event. This gives the worst-case complexity of the algorithm in terms of the number of executing rules per timestep. In practice we expect this number to be very small.

5.4. Distributed Decision Model. To support the autonomous decision making process of each sensor node our approach follows a fully distributed communication model at two levels: local and collective. Local decisions are made based only on the information available in the sensor node while collective decisions require information interchange among the nodes that are involved in the decision. For instance, a data collector decides by itself (i.e., locally) what is the most appropriate light state for the associated lamppost based on its surroundings (see rules 2-3 and 9-10 in Reaction Plan 1). Similarly, a cluster head decides if it should reduce the light level of the lampposts within the cluster based on the information sent from the data collectors (see rule 8 in Reaction Plan 2). Each node fulfills its role by reacting to events in its locally defined manner. While cluster nodes may have the power of authority to modify the light levels of the data collector nodes under its management, the data

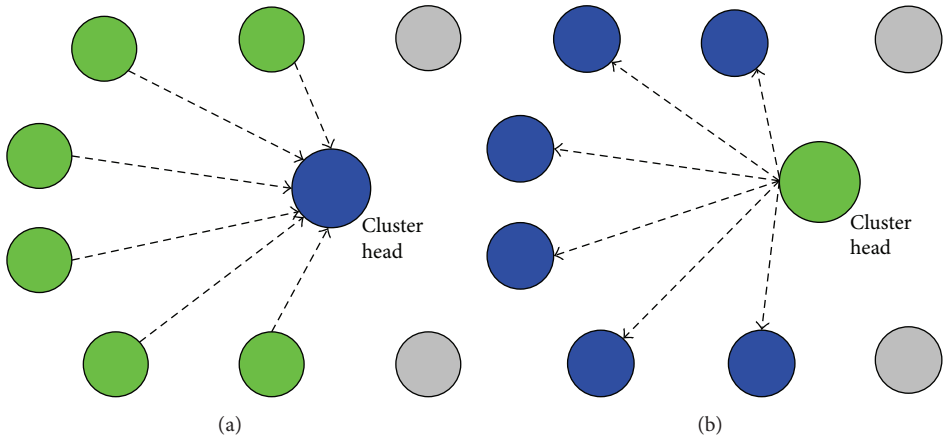

Figure 2: Distributed model at two levels: local (a) and collective (b).

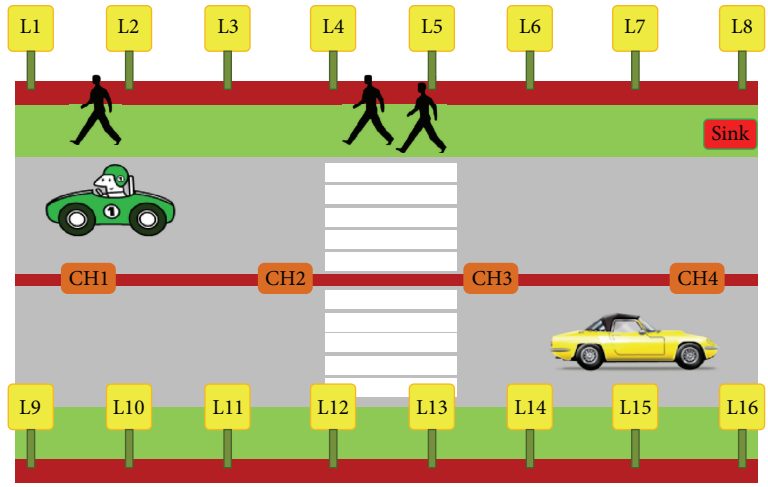

Figure 3: A smart lighting scenario.

collectors have the power of autonomously modifying it based on sensing the local environment. Figure 2 depicts the distributed model used in our approach where decisions are made both locally by the data collectors (on the left hand side) and collectively by the cluster head (on the right hand side) taking into account the messages received from its data collectors. Note that we do not follow a centralized model since we do not have a unique, central, coordinator node which is responsible for the rest of the nodes. The scheme fully distributed is more reasonable to address this type or scenarios, because it allows us to reduce the number of messages required to coordinate the sensor network. In the figure, the sensor nodes that make decisions are represented in green color; gray color indicates that the node is out of the cluster scope.

6. Simulations

We have evaluated the scenario described in Section 5 and pictured in Figure 3 by using SAMSON [27], a framework for WSNs based on strong multiagents in which each sensor node is modeled as a BDI agent. This scenario represents a street with 16 lampposts located on both sides of the street and installed at equal distances from each other. We deploy 21 sensor nodes which play one of the roles described in Section 5: 16 data collectors (L1 to L16), 4 cluster heads (CH1 to CH4), and one sink. Each data collector is attached to one lamppost and each cluster head groups 4 data collectors and routes the messages generated within its cluster towards the sink, which is the device destination of the data. We are interested in validating the use of an agent-based approach as a means of implementing adaptation mechanisms for WSNs and evaluating the energy savings when the adaptation techniques are used. The following subsections present the results.

6.1. Programming Sensor Nodes in AgentSpeak(L). We have programmed the sets of behaviors described in the Reaction Plans 1, 2, and 3 using AgentSpeak(L). As an example, Program 1 shows the encoding of two rules in the Reaction Plan 1, numbers 1 and 11. The first rule `(SAMPLING_TIMER)` represents the process of sampling all the sensors in the data collectors; this behavior is encoded in lines 3–10 in Program 1. The

```
/* Initialgoals */
(1) !start.
/* Plans */
(2) +!start: true <-.at("now + 2s", "+timeToSend").
(3) +timeToSend: true <- +!reading;
            -sensorVal(Env);
            -sensorVal(Solar);
            -sensorVal(Voltage);
            -sensorVal(Presence);
            !start.
(4) +!reading: true <- !senseEnvLight.
(5) +!senseEnvLight: true
            <- sense(2);                                  /* Sensor (2): environmental light */
            !senseSolarLight.
(6) +!senseSolarLight: sensorVal(Env)
            <- sense(5);                                  /* Sensor (5): solar light */
            !addField(Env).
(7) +!addField(Env) : sensorVal(SOLAR)
            <-.concat(SOLAR, "->", Env, Msg);
            !senseVoltage(Msg).
(8) +!senseVoltage(Msg): true
            <- sense(4);                                  /* Sensor (4): voltage */
            !sensePresence(Msg).
(9) +!sensePresence(Msg): sensorVal(Voltage)
            <- sense(6);                                  /* Sensor (6): presence */
            .concat(Voltage, "->", Msg, SendingMsg);
            !send(SendingMsg).
(10) +!send(SendingMsg): sensorVal(Presence)
            <-.concat(Presence, "->", SendingMsg, message);
            setMCUState(0);                               /* MCU on */
            setRadioState(3);                             /* Radio on */
            setTXPower(-24);
            sendTX("sensor","cluster_head",message);
            setRadioState(2);                             /* Radio off */
            setMCUState(2).                               /* MCU standby */
```

PROGRAM 1: Fragment of code in AgentSpeak(L) for the nonadaptive (non-A) behavior of data collectors.

second rule (SENDING_DATA) sends a message containing the sensor measurements to its cluster head; analogously, this behavior is codified in line 10 in Program 1. We consider sampling and sending intervals of 2 s each. This simple behavior does not allow the lampposts to adapt the intensity of the light they produce to the actual conditions; we call this the nonadaptive (non-A) behavior.

Program 1 behaves as follows: it starts by adding the initial achievement goal `!start` to the agent's beliefs. This goal generates an internal action, called.at, which schedules a timer to go off after 2 s. When the event `!timeToSend` is triggered (which will happen approximately in two seconds), the actions in the corresponding plan body are scheduled to execute: firstly the achievement goal `!reading`, followed by the deletion of the beliefs of type `-sensorVal`, and finally the achievement goal `!start`, which reinitiates the timer. The deletion of the beliefs gets rid of the old measurements. Note that, due to resource limitations, the reading of the transducers on a sensor must be done sequentially; it is therefore incorrect to start the reading of a sensor while the previous reading is still ongoing. To save the results of the readings from one behavior to another we concatenate these values (via the internal action.concat) into a variable that is passed as an event parameter. `!reading` generates an event that triggers the execution of the goal `!senseEnvLight`. This event triggers the reading of the environment light sensor (identified by number 2 (The numbering used for the sensors depends on the implementation done for TMoteSky.)) and adds a new goal `!senseSolarLight`—which continues the reading of the rest of the transducers. Note that when `!senseSolarLight` is triggered, its plan first checks that the environmental light data is available. Only then the new achievement goal `!addField` is scheduled. `!addField` checks that the value of the solar light transducer is available and then concatenates the solar and the environment light data into the variable *Msg*. Finally, it schedules the achievement goal `!senseVoltage`. This process is repeated until all the sensor values are available, at which point the goal `!send` is added for execution in rule 9. The triggering of this event invokes several Java functions which turn on the radio, send the data in *SendingMsg* to the cluster head, and turn the radio off.

A few comments are in order. First, the radio is turned off after sending the message to the cluster head by means of `setRadioState(2)`, which results in saving some energy compared to the case in which the radio is left on. It is this energy-efficient behavior that we compare our adaptive approach against. Second, TMote Sky does not include some of the sensors that we use in our scenario. Instead the platform incorporates transducers for temperature, humidity, TSR/PAR light, and voltage. We added support to SAMSON for simulating transducers for solar light and object presence.

6.2. Simulation of the Adaptive Application. To validate the adaptive application we first modified SAMSON to adapt it to our requirements. The main additional functionality includes (1) adding sensors used by the application but inexistent on the TMote Sky platform and (2) establishing a network deployment that fits the scenario that we are simulating. Functionality (1) implied modifying the SAMSON source code to include support for `SOLAR_LIGHT` and `PRESENCE_SENSOR` sensors as well as for `LIGHT_LAMP` actuator; we used the sensor TSR of the TMote Sky platform as `ENV_LIGHT` sensor. Functionality (2) is not provided by SAMSON, which deploys the sensor nodes randomly on a square. In our scenario the data collectors and cluster heads have specific positions which capture the real geographic configuration. We add the possibility of placing the sensor nodes at fixed positions within the square according to a given deployment. Figure 4 shows the agent-based WSN to be evaluated in SAMSON, which reproduces the location of the data collectors (lampposts), cluster heads, and the sink that we introduced in our scenario of Figure 3. This figure is also showing the transmission of a message from CH4 and its reception by all the sensors within its neighborhood: CH3, L6–L8, and L14–L16. The red color indicates that a sensor is transmitting a message; the orange color indicates that a sensor receives it. The propagation wave is represented as a blue circle centered in the transmitter node. Note that the routing done by CH2 and CH3 towards the sink is compulsory for the messages proceeding from the farthest data collectors to arrive at their destination (the sink).

We implemented the behaviors described in Reaction Plan 1 using AgentSpeak(L); we call this the adaptive application (A) (see Figure 1 to remember the transitions among the light levels in the (A) application). The purpose of the adaptive application is to reduce the energy consumption per lamppost. By reducing the time spent in states with higher light intensity levels we can reduce the energy consumption. Our simulations are mainly directed towards evaluating the fraction of the simulation time that each lamppost spends in each state (`OFF`, `HIGH`, `MEDIUM`, and `LOW`) and using these times to estimate its energy consumption and the energy savings relative to the nonadaptive application. On the scenario presented in Figure 3 we simulate two different use cases: (1) a residential street where there is no additional environment light and there is a low frequency of pedestrians, and (2) a commercial street with high frequency of pedestrians and on which 50% of the lampposts detect environment light. To model the presence of pedestrians we implemented a simple

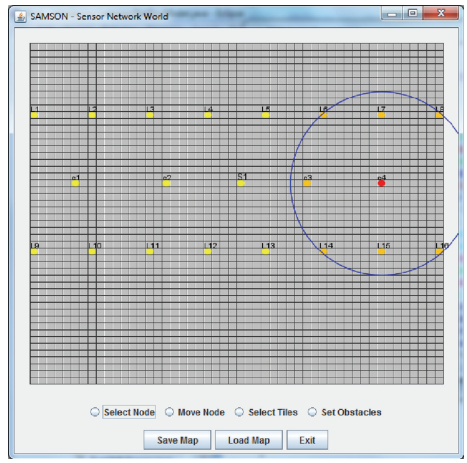

Figure 4: A window of SAMSON showing a topology of agents located at fixed positions within the square.

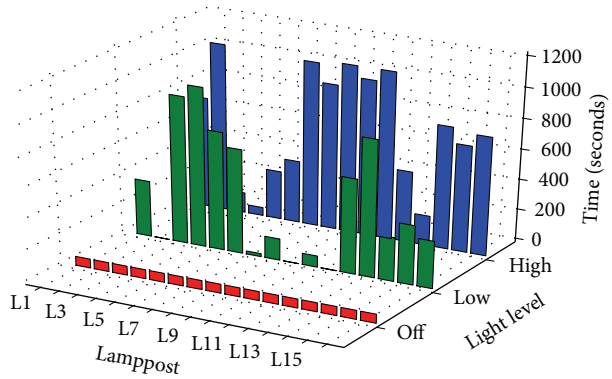

Figure 5: Results for (A) application for the residential street use case.

presence sensor which behaves as follows. The sensor returns one of two possible values: 0 for absence of movement and 1 for its presence. As Section 5 describes, we are only interested in objects that present a certain degree of persistence in the vicinity of a lamppost, that is, those who maintain their location during some minimum period of time. We impose this restriction to avoid a behavior of the lamppost which switches on and off immediately when some object is sensed, and which could lead to an additional waste of energy.

We simulate the residential street use case for approximately 19.3 minutes (1158 s) and we measure the time spent by each lamppost in each of the four states. Figure 5 shows the times (in seconds) per state for each lamppost. 60 seconds after starting the simulation (when nighttime is detected), all the lampposts transit from `OFF` to `HIGH`. Depending on when (or whether) object presence is detected, most of the lampposts (except for L2, L9, and L11) eventually transit to state `LOW`. In this use case there is no transition to the `MEDIUM` state since we assume that environment light is not detected. SAMSON also reports some information about the sensor nodes, in particular the voltage and the number of packets transmitted (Tx) and received (Rx) by each node. Figure 6 on

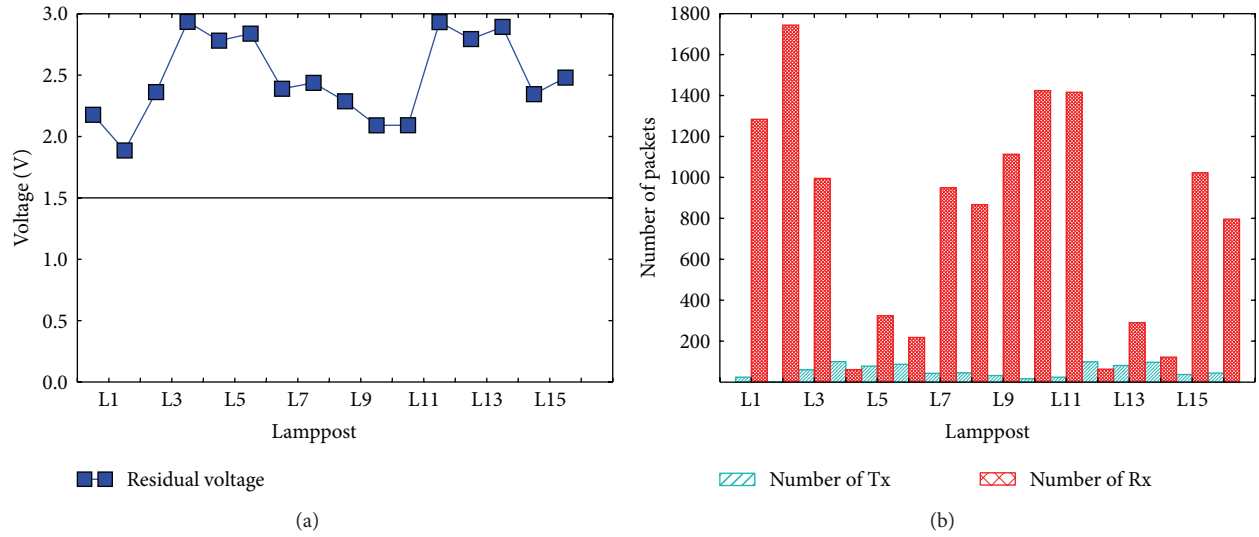

FIGURE 6: Residual voltage after simulation (a) and number of packets transmitted and received (b) per lamppost in a residential street.

the left shows the value of the voltage for the data collectors, which is maintained relatively high—between 1.9 and 2.8 V; on the right side we show the number of packets transmitted and received. If the voltage is lower than 1.5 V the node stops its operation and dies. We observe that the number of packets sent differs for each data collector due to the fact that the simulator starts its operations at different times. For each node, the number of received packets include all those transmitted by the nodes in the neighborhood, although possibly to a different destination, in such a way that the Rx packets vary depending on the location of the node within the topology. A data collector does not retransmit the data packets received from its neighbors. We also observe that the voltage is related to the number of packets transmitted and received; the sum of both is inversely proportional to the residual voltage in the sensor node.

Now, we simulate the commercial street use case for approximately 82 minutes (4920 s). We consider that a group of lampposts obtain values for the environment light that are above the given threshold. This may be the case if lampposts L1–L8 are located on the side of the commercial street which has shops while lampposts L9–L16 are on the dark side. This implies that lampposts L1–L8 transit from state `HIGH` or `LOW` to `MEDIUM` when they detect the presence of an object, but lampposts L9–L16 cannot transit into `MEDIUM` because they sense no additional light and therefore will remain in state HIGH in the presence of an object. Any lamppost which stops detecting object presence transits to state `LOW`. When presence is detected again L1–L8 returns to state `MEDIUM` and L9–L16 returns to `HIGH`. Figure 7 shows this behavior. A lamppost with no adaptability must remain in state `HIGH` overnight. With adaptation, the lampposts adjust their state to the changes in the environment to remain in the most appropriate state and waste less energy. Figure 8 on the left shows the residual voltage in the sensor node; on the right we show the number of packets transmitted and received.

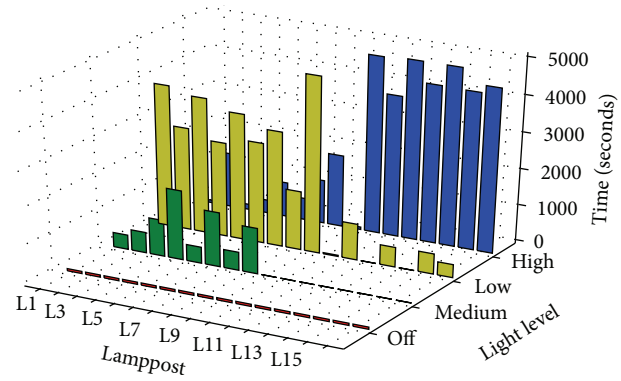

FIGURE 7: Results for (A) application for the commercial street use case.

6.3. Energy Saving Estimation for Street Lighting. Assuming that the lampposts use LED technology and knowing (from the simulations) the percentage of time that each of them spends in each state, we can estimate the energy consumption (in joules) due to the emission of light as $E(J) = \sum_{i=1\cdots s} t_i \times V \times I_i$, where s is the light intensity state ($s = \{\texttt{OFF}, \texttt{HIGH}, \texttt{MEDIUM}, \texttt{LOW}\}$), t_i is the time in state $i \in s$, V is the voltage required for the device to work properly, and I_i is the current draw to emit the light corresponding to state $i \in s$. We do not consider additional energy consumption sources because the information that is available only has to do with the time spent in the four intensity states. For any state i that represents a greater intensity level than that of a state j, the energy consumption in state i will be bigger than in state j. It follows that $I_{\text{off}} < I_{\text{low}} < I_{\text{medium}} < I_{\text{high}}$, which means that the energy consumption is reduced when a lamppost spends less time in a state of high intensity. The energy consumption of the LEDs depends on several factors such as the color of the light, the manufacturer, and the power to be supplied. However, a

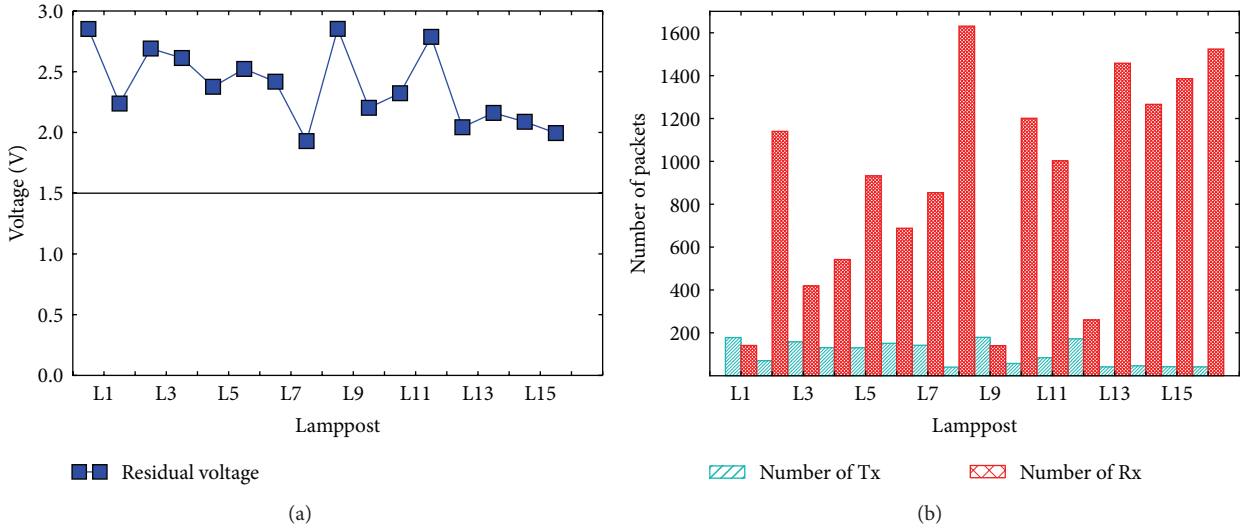

Figure 8: Residual voltage after simulation (a) and number of packets transmitted and received (b) per lamppost in a commercial street.

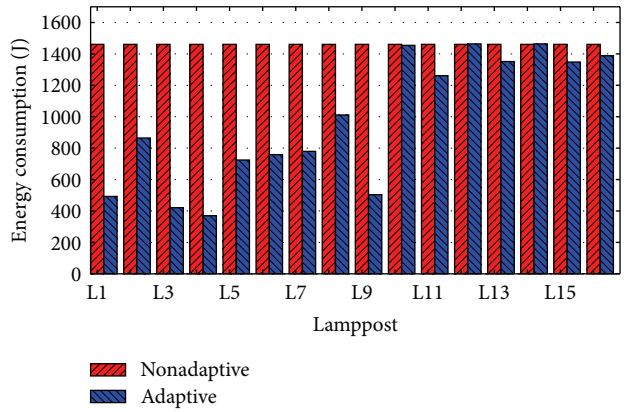

Figure 9: Comparison between the energy consumed by the lampposts executing the (non-A) versus (A) applications.

consumption between 15 and 100 milliamperes is normally assumed. The voltage ranges between 3 and 12 V.

Let $\omega = 100\,\text{mA}$ be the maximum current draw for a LED. We estimate a current draw for the different states of a LED to be: $I_{\text{high}} = \omega$, $I_{\text{medium}} = (2/3)\,\omega$, $I_{\text{low}} = \omega/3$, and $I_{\text{off}} = \omega/6$. For simplicity sake we assume that each lamppost in our scenario consists of a single LED. We compute the energy consumed by all lampposts when they execute our nonadaptive application (non-A) versus the adaptive application (A); each simulation runs for 4920 seconds. For (non-A) the lampposts transit from `OFF` to `HIGH` after 60 seconds of simulation and remain in this state for the rest of the simulation. For (A) we consider the time spent in each state as presented in Figure 8. The comparison is shown in Figure 9. As expected, most of the lampposts save energy when implementing the adaptive application; the larger savings occur for L1–L8, which encounter favorable conditions in the form of environment light.

Energy savings result in cost savings. To analyze this effect we consider a real LED-based lamp, the LS8 [31] lamp manufactured by Build Better Earth. According to its datasheet, the LS8 provides a maximum rated power of 240 W. We assign the power values for the rest of the intensity states—$P_{\text{high}} = 240\,\text{W}$, $P_{\text{medium}} = 240/2 = 120\,\text{W}$, $P_{\text{low}} = 240/4 = 60\,\text{W}$, and $P_{\text{off}} = 0$—and we compute the energy savings assuming the average time per hour spent in each state for all lampposts. The results are shown in Table 1, where we take the cost of a kilowatt per hour to be 15 cents of euro. As observed, our estimation is that the cost saving per night (10 hours) is approximately 35%.

6.4. Simulation of a Real Smart-City Scenario. We conclude our experiments by simulating the adaptive application on a real city data—Leganés, located at the south of Madrid (Spain)—where approximately 50,000 lampposts exist. During winter nights the lampposts are turned on in state `LOW` at 6:00 pm and progressively increase the intensity to reach the `HIGH` state at 7:00 pm. They remain in this state until 5:00 am, when they progressively decrease their intensity to reach the state `OFF` at 7:00 am. The left side of Figure 10 shows the transitions among these states at the time they happen.

Following estimations by the city council of Leganés, the average probability for the presence of people in residential areas during winter nights is 50% from 8:00 pm to 12:00 am, 10% from 12:00 am to 4:00 am, and 40% from 4:00 am to 5:00 am. We evaluate the effect of these probabilities on energy savings in Leganés when assuming that the lampposts implement our adaptive mechanism. To do this, we fix a simulation time of 2000 s and force transitions among each two intensity states after a period of time proportional to the real-time transitions prescribed by the city. The resulting time spent in each state is shown in Figure 11. We observe the lampposts changing their intensity level according to the detection of the presence of people and the existence of environment light, as opposed to the nonadaptive behavior. This adaptive behavior is shown on the right hand side of Figure 10 for a specific lamppost, L15. During the time frame with the

Table 1: Estimation of the energy costs per lamppost, overnight and per year, for the (non-A) and (A) applications.

State	Rated power (W)	Average time per hour	Adaptive application		Nonadaptive application	
			Power per hour (W)	Cost per night (€)	Power hour (W)	Cost per night (€)
High	240	0.49	117.83	0.176	240	0.36
Medium	120	0.11	13.20	0.0198	0	0
Low	60	0.40	24.01	0.036	0	0
Total lamppost/night (€)			0.23		0.36	
Total lamppost/year (€)			**83.95**		**131.4**	

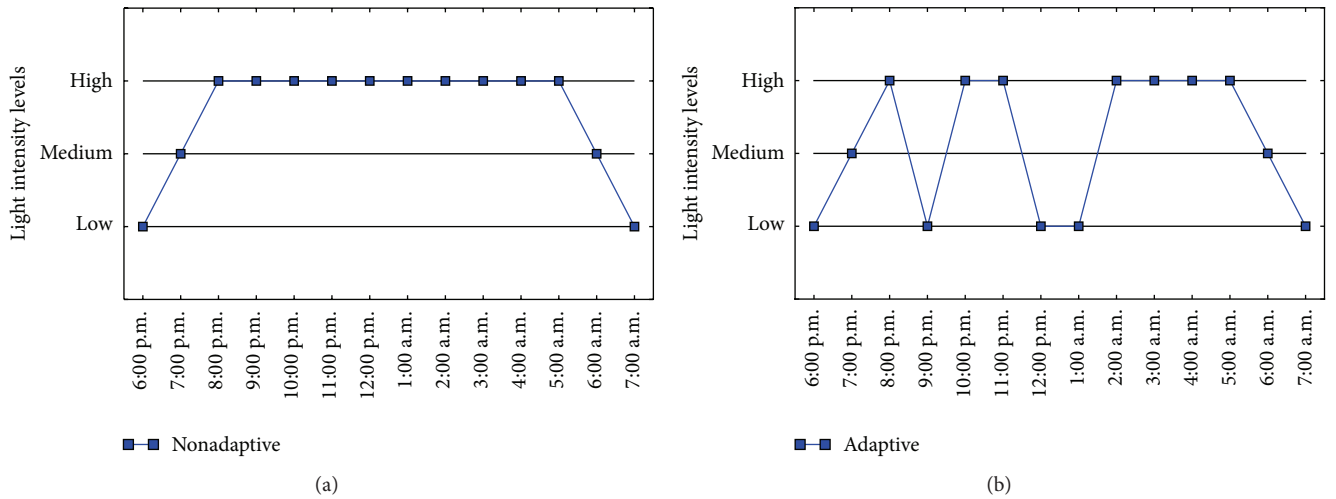

Figure 10: Behavior of the lampposts in Leganés during winter nights: on the left, the nonadaptive behavior for all lampposts; on the right, the adaptive behavior for a specific lamppost, L15.

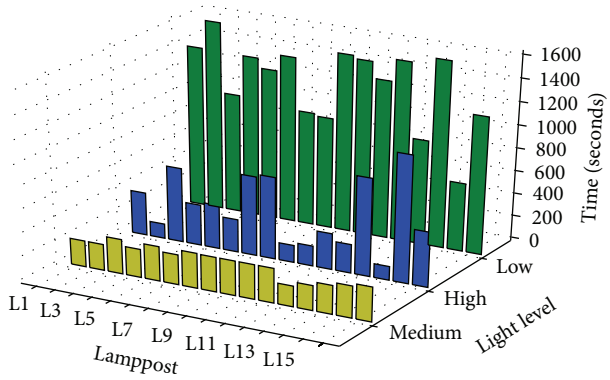

Figure 11: Results for (A) application in the city of Leganés.

lowest probability of detecting people in the street (12:00 am–4:00 am) it is most probable that lampposts may transit to state `LOW`, although they will not turn off completely to reduce the insecurity of the citizens. We observe that the state `MEDIUM` is reached only twice, once at the beginning and once at the end of the night. This behavior is similar to that in the nonadaptive application since we assume that in residential areas the environment light is not strong enough to surpass the threshold and transit the lampposts into the `MEDIUM` state. Figure 12 shows the voltage and packet numbers for this simulation as reported by SAMSON.

Finally, Table 2 shows the cost savings when implementing adaptive behavior in this real city scenario. The energy savings reach 55% relative to the nonadaptive application.

7. Conclusions

This paper presents an intelligent street light control system that enables multiphase light sources to adapt their intensity to the environment conditions. We have designed an adaptive behavior for the control devices attached to the lampposts in smart cities scenarios; as a result the lampposts dynamically adapt to the presence (or absence) of obstacles and environment light in their vicinity. These are only two examples of the data that could be monitored and used for the fine tuning of the services provided by a smart city in a way that promotes sustainability.

We have evaluated our approach by using a simulator that combines WSNs and BDI agents and provides information about the time that each lamppost spends in each intensity state. Starting from these times we estimate the energy savings when compared to nonadaptive approaches. The results show important savings above 35% in all the experiments; these have a significant economic impact and affect the sustainability of the environment.

As future work we plan to exploit our approach on more dynamic scenarios, where real-time data—for instance, proceeding from Google Traffic—can be used to evaluate the

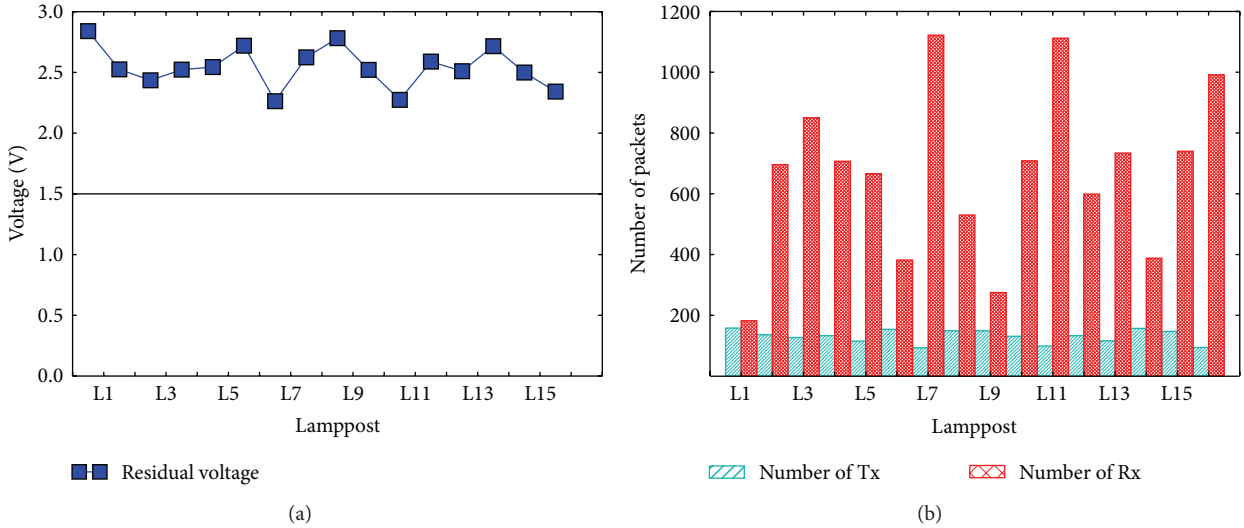

Figure 12: Residual voltage after simulation (a) and number of packets transmitted and received (b) per lamppost in the city of Leganés.

Table 2: Estimation of the energy costs per lamppost, overnight and per year, for the city of Leganés.

State	Rated power (W)	Average time per hour	Adaptive application		Nonadaptive application	
			Power per hour (W)	Cost per night (€)	Power hour (W)	Cost per night (€)
High	240	0.22	53.33	0.079	240	0.36
Medium	120	0.14	16.38	0.024	0	0
Low	60	0.64	38.49	0.057	0	0
Total lamppost/night (€)			0.162		0.36	
Total lamppost/year (€)			**59.13**		**131.4**	

adaptive behavior of the devices. We also plan to include more sophisticated algorithms which conserve sensor energy along with street light energy.

Conflict of Interests

The authors declare that he has no conflict of interests regarding the publication of this paper.

Acknowledgment

This work has been funded by the Spanish Ministry of Science and Technology under the Grant TIN2010-16497 "Técnicas Escalables de E/S en Entornos Distribuidos y de Computación de Altas Prestaciones."

References

[1] G. S. Dutt, "Illumination and sustainable development. Part I: technology and economics," *Energy for Sustainable Development*, vol. 1, no. 1, pp. 23–35, 1994.

[2] M. Stockman, "Light-emitting devices: from nano-optics to street lights," *Nature Materials*, vol. 3, no. 7, pp. 423–424, 2004.

[3] W. Murff and Kuntz, "Winter 2011: sustainable cities: Salem street lights," City of Salem, February 2012, http://www.cityofsalem.net/Residents/Sustainable-Salem/SCI/Documents/StreetLights/Group1_Book.pdf.

[4] P. Leonard, "Memorandum: street lights," City of Dublin, February 2010, http://www.upperdublin.net/information/sustainable/streetlights.aspx.

[5] L. Martirano, "A smart lighting control to save energy," in *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS '11)*, vol. 1, pp. 132–138, September 2011.

[6] P. Elejoste, I. Angulo, A. Perallos et al., "An easy to deploy street light control system based on wireless communication and led technology," *Sensors*, vol. 13, no. 5, pp. 6492–6523, 2013.

[7] T. P. Huynh, Y. K. Tan, and K. J. Tseng, "Energy-aware wireless sensor network with ambient intelligence for smart LED lighting system control," in *Proceedings of the 37th Annual Conference of the IEEE Industrial Electronics Society (IECON '11)*, pp. 2923–2928, November 2011.

[8] X. Cao, J. Chen, Y. Xiao, and Y. Sun, "Building-environment control with wireless sensor and actuator networks: centralized versus distributed," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 11, pp. 3596–3605, 2010.

[9] J. S. Sandhu, "Wireless sensor networks for commercial lighting control: decision making with multi-agent systems," in *Proceedings of the AAAI Workshop on Sensor Networks*, pp. 131–140, 2004.

[10] Y. Wu, C. Shi, X. Zhang, and W. Yang, "Design of new intelligent street light control system," in *Proceedings of the 8th IEEE International Conference on Control and Automation (ICCA '10)*, pp. 1423–1427, June 2010.

[11] L. Y. X. Jun and P. Yonglong, "Study of energy-saving solar street light using led based on mcu-controlled," *Test & Measurement Technology*, no. 10, pp. 29–31, 2008.

[12] IERC—Internet of Things, "The Internet of Things 2012—New Horizons," 2012, http://www.internet-of-things-research.eu/pdf/IERC_Cluster_Book_2012_WEB.pdf.

[13] M. Weiser, "The computer for the 21st century," in *Human-Computer Interaction*, R. M. Baecker, J. Grudin, W. A. S. Buxton, and S. Greenberg, Eds., pp. 933–940, Morgan Kaufmann, San Francisco, Calif, USA, 1995.

[14] R. K. Bose and G. Anandalingam, "Sustainable urban energy-environment management with multiple objectives," *Energy*, vol. 21, no. 4, pp. 305–318, 1996.

[15] E. K. Zavadskas, A. Kaklauskas, and J. Kaklauskiene, "Modelling and forecasting of a rational and sustainable development of Vilnius: emphasis on pollution," *International Journal of Environment and Pollution*, vol. 30, no. 3-4, pp. 485–500, 2007.

[16] Y. R. Jabareen, "Sustainable urban forms: their typologies, models, and concepts," *Journal of Planning Education and Research*, vol. 26, no. 1, pp. 38–52, 2006.

[17] M. Vinyals, J. A. Rodriguez-Aguilar, and J. Cerquides, "A survey on sensor networks from a multiagent perspective," *The Computer Journal*, vol. 54, no. 3, pp. 455–470, 2011.

[18] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 4–50, 2003.

[19] K. Schneider, T. Schuele, and M. Trapp, "Verifying the adaptation behavior of embedded systems," in *Proceedings of the International Workshop on Self-Adaptation and Self-Managing Systems (SEAMS '06)*, pp. 16–22, ACM, 2006.

[20] K. Hazelwood, "Process-level virtualization for runtime adaptation of embedded software," in *Proceedings of the 48th Design Automation Conference (DAC '11)*, pp. 895–900, ACM, 2011.

[21] J. Cámara, G. Salaün, C. Canal, and M. Ouederni, "Interactive specification and verification of behavioral adaptation contracts," *Information and Software Technology*, vol. 54, no. 7, pp. 701–723, 2012.

[22] F. Fleurey, B. Morin, and A. Solberg, "A model-driven approach to develop adaptive firmwares," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '11)*, pp. 168–177, ACM, 2011.

[23] N. R. Jennings, "On agent-based software engineering," *Artificial Intelligence*, vol. 117, no. 2, pp. 277–296, 2000.

[24] A. S. Rao and M. P. Georgeff, "Bdi agents: from theory to practice," in *Proceedings of the 1st International Conference on Multiagent Systems (ICMAS '95)*, pp. 312–319, 1995.

[25] A. S. Rao, "AgentSpeak(L): BDI agents speak out in a logical computable language," in *Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW '96)*, pp. 42–55, Springer, Eindhoven, The Netherlands, January 1996.

[26] R. H. Bordini and J. F. Hbner, "Bdi agent programming in agentspeak using jason," in *Proceedings of the 6th International Workshop on Computational Logic for Multi-Agent Systems (CLIMA VI '05)*, vol. 3900 of *Lecture Notes in Computer Science*, pp. 143–164, Springer, 2005.

[27] A. Morris, *SAMSON: strong multi-agent simulation of wireless sensor networks [Ph.D. dissertation]*, University College Dublin, Dublin, Ireland, 2008.

[28] M. Corporation, "Tmote sky. Low Power Wireless Sensor Module," 2006, http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf.

[29] S. Escolar, S. Chessa, J. Carretero, and M. C. Marinescu, "Cross layer adaptation of check intervals in low power listening mac protocols for lifetime improvement in wireless sensor networks," *Sensors*, vol. 12, no. 8, pp. 511–510, 2012.

[30] S. Escolar, S. Chessa, and J. Carretero, "Energy management in solar cells powered wireless sensor networks for quality of service optimization," *Personal and Ubiquitous Computing*, 2013.

[31] B. B. Earth, "Ls led street lights—ls8," June 2013, http://www.bbeled.com/downloads/ls8-spec-vo.pdf.